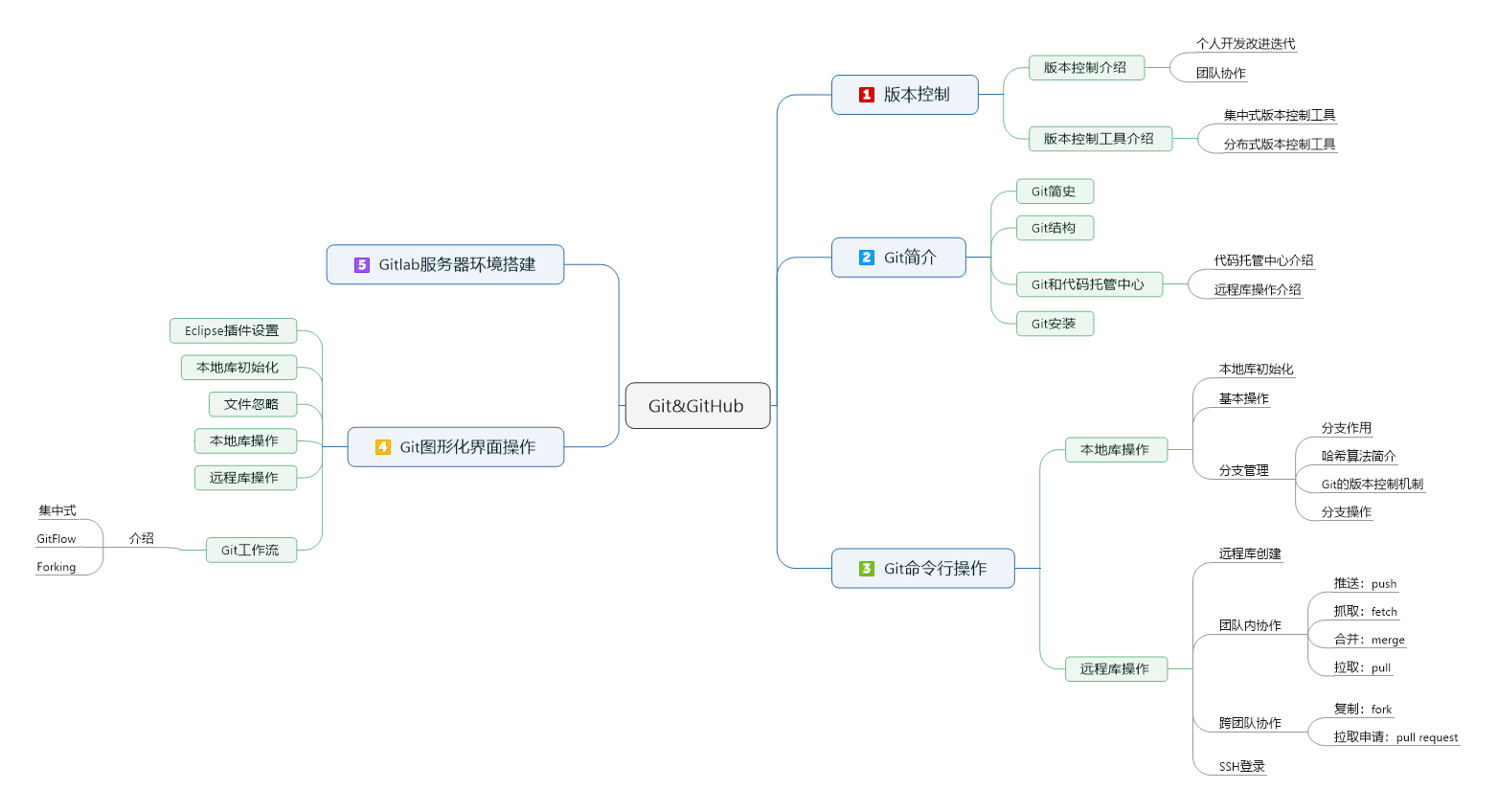
# Git&GitHub



## 版本控制

### 1.版本控制工具应该具备的功能

#### 1.协同修改  多人能够并行不悖的修改服务器端的同一个文件。 

#### 2.数据备份  不仅保存目录和文件的当前状态，还能够保存每一个提交过的历史状态。 

#### 3.版本管理  在保存版本的文件信息时要做到不保存重复数据以节约存储空间，提高运行效率。SVN 采用的是增量式管理的方式，而 Git 采取了文件系统快照的方式。 

#### 4.权限控制  对团队中参与开发的人员进行权限控制。对团队外开发者贡献的代码进行审核（Git独有）。 

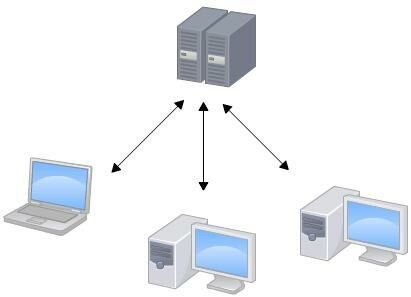
#### 5.历史记录  查看修改人、修改时间、修改内容、日志信息。将本地文件恢复到某一个历史状态。 

#### 6.分支管理  允许开发团队在工作过程中多条生产线同时推进任务，进一步提高效率。

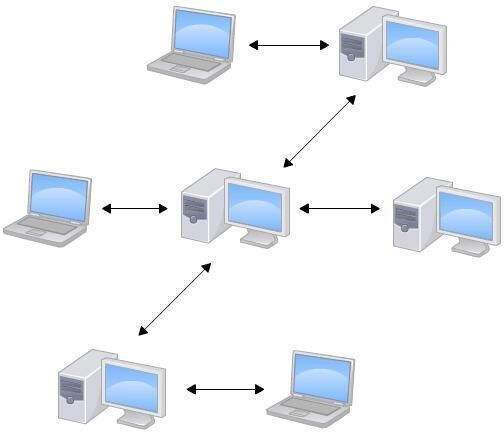
### 2.版本控制工具就是版本控制思想的实现

工程设计领域中使用版本控制管理工程蓝图的设计过程。在IT开发过程中也可以使用版本控制思想管理代码的版本迭代。

#### 1.集中式版本控制工具：CVS、SVN、VSS…

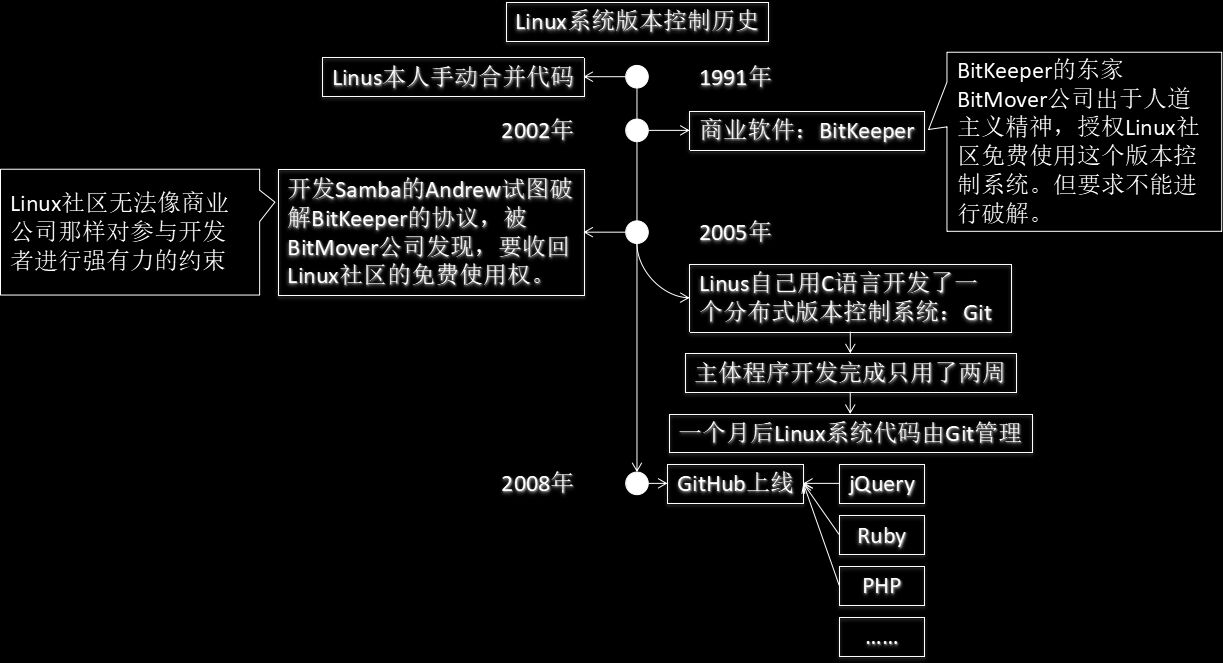


#### 2.分布式版本控制工具：Git、Mercurial、Bazaar、Darcs…



## Git简介

### 1.简史

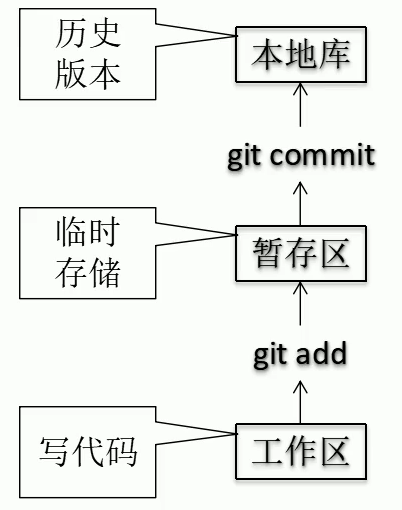
  https://git-scm.com/

### 2.Git的优势

1. 大部分操作在本地完成，不需要联网
2. 完整性保证（hash）
3. 尽可能添加数据而不是删除或修改数据
4. 分支操作非常快捷流畅
5. 与 Linux 命令全面兼容

Tips:Git安装时一致点下一步即可，建议在调整环境变量时选择最安全的方式

### 3.Git结构



### 4.Git和代码托管中心

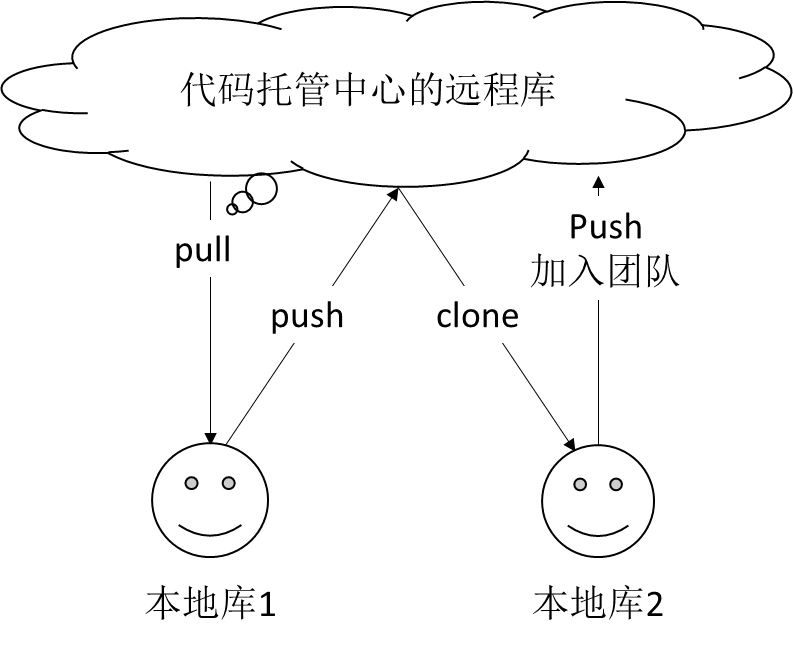
代码托管中心的任务：维护远程库

局域网环境下：GitLab 服务器

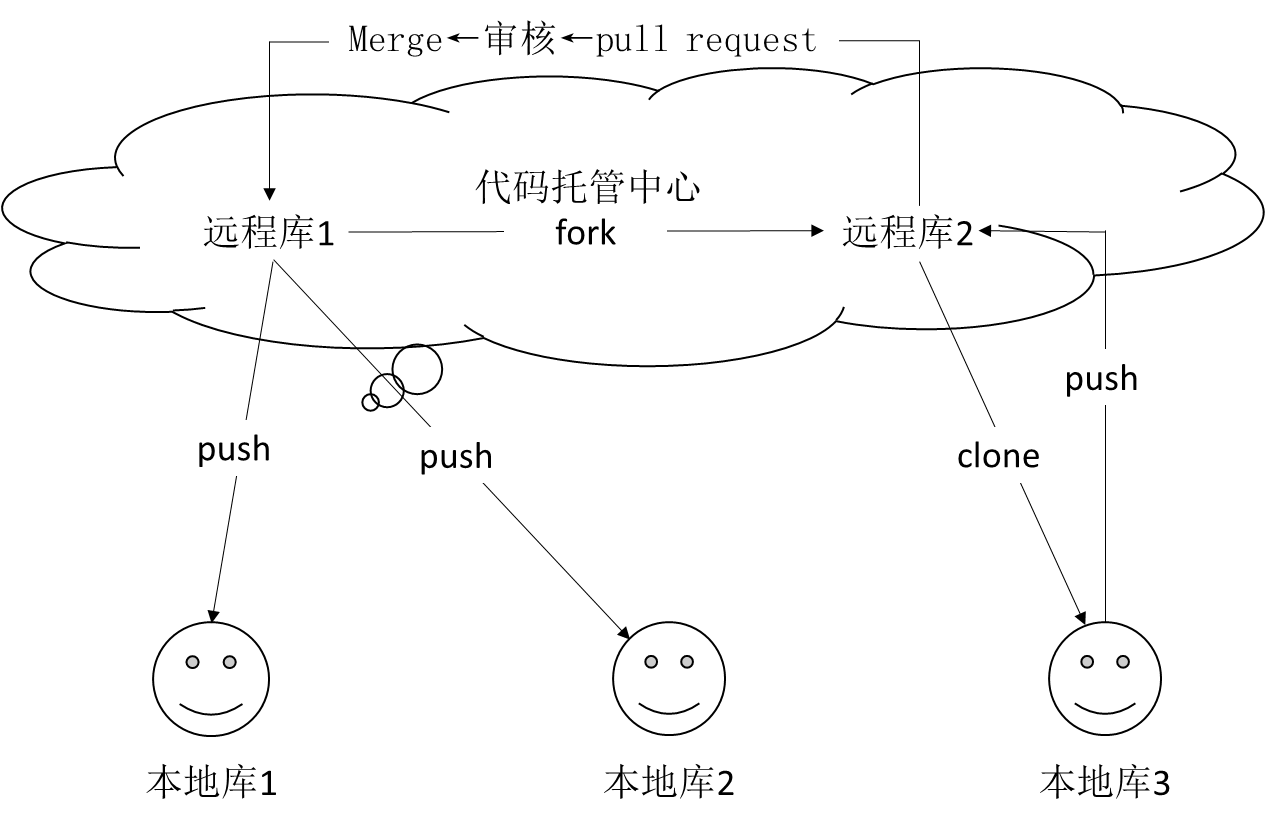
外网环境下：GitHub（外国，速度慢）、码云

### 5.本地库和远程库

#### 1.团队内部协作



#### 2. 跨团队协作



### 6.Git命令行操作

Git命令与Linux命令兼容

#### 1.本地库初始化

进入创建的本地库目录中执行 git init ，这时会创建 .git 隐藏目录，其中存放了与本地库相关的子目录和文件

#### 2.设置签名

作用：区分不同开发人员的身份

辨析：这里设置的签名和登录远程库（代码托管中心）的账号、密码没有任何关系。

命令：

项目级别/仓库级别：仅在当前本地库范围内有效，信息保存在….git/config文件中

git config user.name yourname

git config user.email youremail

系统用户级别：登录当前操作系统的用户范围，信息保存在 ~/.gitconf文件中

git config --global yourname

git config --global youreemail

两者必须存在一个，系统用户级别优先级较低

#### 3.基本操作

1.git status 查看工作区、暂存区状态

2.git add file 将工作去的添加或修改添加到暂存区，如果是修改文件，不想放到暂存区，可以不执行本命令

3.git commit（-m “message”）file 将暂存区的文件提交到本地库，不加括号内容则会到文件中内容则到文件中添加提交信息

4. git rm --cached file 从暂存区移除文件

5.git log（--pretty=oneline） 查看文件提交的日志信息，git log --oneline更简洁，文件太多时space可向下翻页，p向上，q退出

6.git reflon 显示日志的同时显示HEAD指针回归版本需要的移动步数

7.git reset --hard索引值 修改当前文件版本，一旦执行commit就会添加新的版本文件，这些文件不会被删除，除非删除本地库

git reset --hard HEAD ^… 只能用来后退版本，几个^符号代表后退几步，不添加^就回到当前指针指向的版本

git reset --hard HEAD~n 也只能用来后退版本，n指定步数

reset命令的三个参数对比

--hard 移动HEAD指针、重置暂存区、重置工作区，可通过这种方式来找回文件

--mixed 移动HEAD指针、重置暂存区

--soft 移动HEAD指针

8.git help 命令 打开本地命令帮助文档文件查看命令提示

9.git diff（本地库版本文件）file 将工作区文件与历史最新版本做比较。添加扩号内容为与指定版本比较，可以用^和~。不指定文件名则比较工作区所有文件

10.git branch branchname 创建分支

git branch branch –v 查看分支

git checkout branchname 切换分支

git merge branchname 合并分支，需要先切换到旧内容的分支执行

当两个分支同时修改了一个文件时，会出现合并冲突，这时冲突文件会出现这样的内容：

<<<<<<< HEAD 解决步骤：

当前分支内容

======= 1.编译文件，删除特殊符号，决定正确的版本

合并分支内容 2.git add filename

>>>>>>> master 3.git commit （-m “message”），这里不带文件名

## Git基本原理

### 1.哈希

哈希是一系列的加密算法，不同的哈希算法虽然加密强度不同，但是有几个共同点：

1.不同大小的不同文件执行同一个哈希算法，输出结果的长度固定。

2.相同输入数据执行同一个哈希算法，输出结果不变

3.输入数据稍有变化，输出结果变化巨大

4哈希算法不可逆

Git底层采用的是 SHA-1 哈希算法，通过这种hash机制从根本上来保证数据的完整性

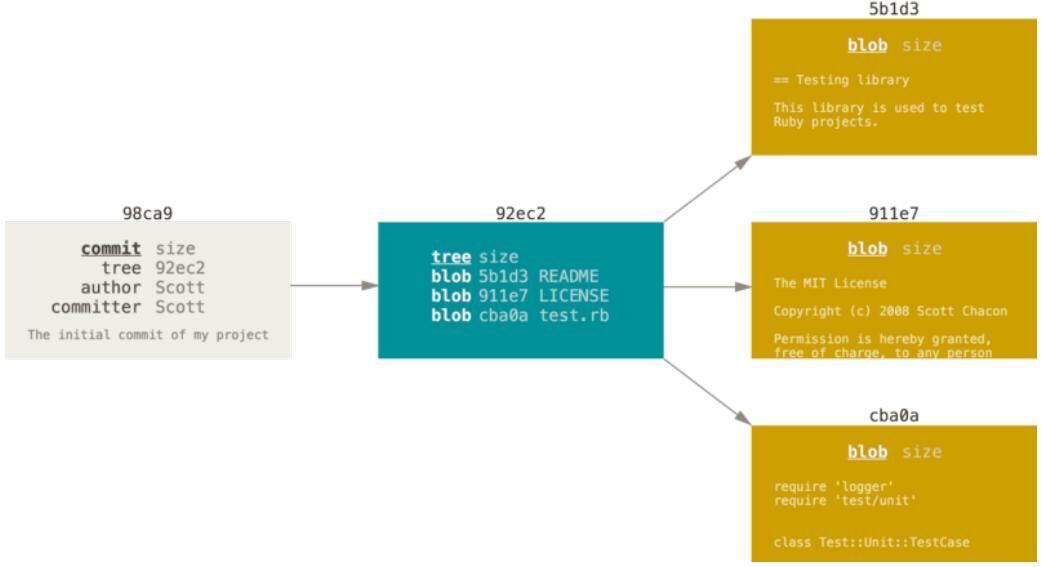
### 2.Git保存版本的机制

#### 1.集中式版本控制工具的文件管理机制

以文件变更列表的方式存储信息，每个版本保存的是变更的内容。这类系统将保存的信息看作是一组文件和文件随时间逐步累积的差异，获取文件时将一组文件的信息拼接后返 回，能够节约存储空间。

#### 2.Git的文件管理机制

Git把数据看作是小型文件系统的一组快照。每次提交更新时，Git都会对当前的全部文件制作一个快照并保存这个快照的索引。为了高效，如果文件没有修改，Git 不再重新存 储该文件，而是只保留一个链接指向之前存储的文件。



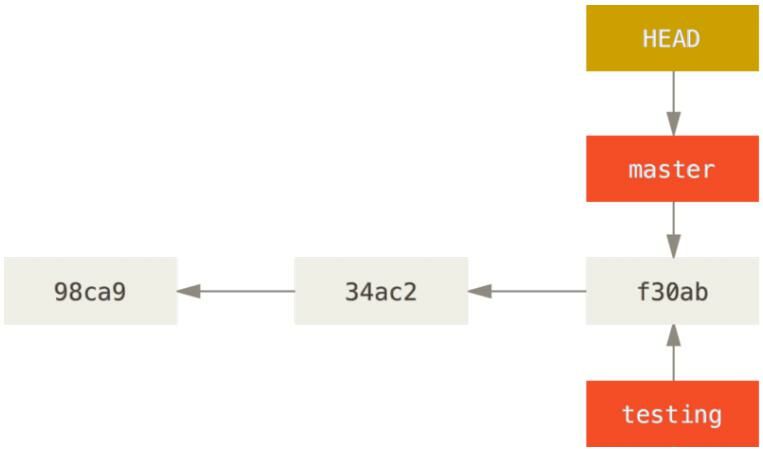
#### 3.Git文件管理机制细节

1. 文件对象：提交文件，有hash值
2. 树对象：提交目录，有hash值，包含每个具体的提交对象
3. 提交对象：也有hash值，包含树对象和提交信息，

比如时间，修改者，父对象之类，父对象就是上一个版本

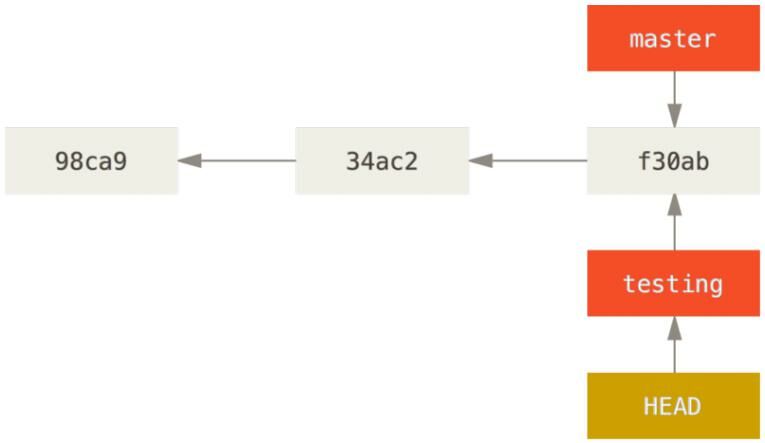
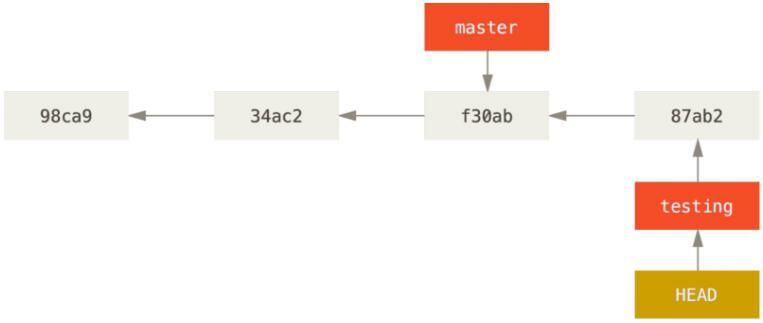
#### 4. Git分支管理机制

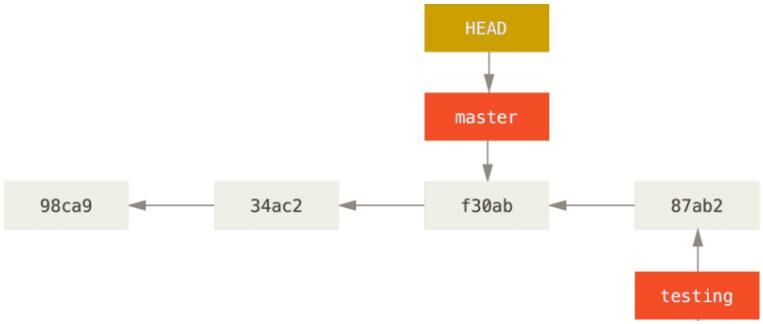
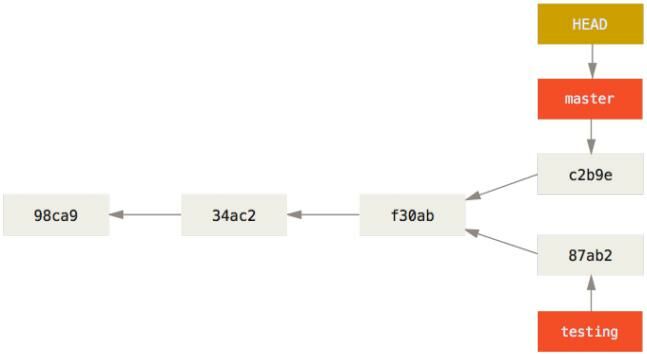
1.分支的创建

 分支也相当HEAD指针，Git创建分支就相当于创建一个指针，而不是cope一份数据

root commitoot

2.分支的切换

## GitHub

1.创建远程库别名 git remote add alias <https://github.com/15308232110/MyFirestRepository.git> git remote –v 为查看

2.推送 git push alias branchname

3.克隆 git clone <https://github.com/15308232110/MyFirestRepository.git> 三个作用：1.拷贝远程库 2.初始化本地库 4.可以实现创建别名

4.授予push权限 进入远程库 → Setting → Collaborators → 输入被邀请人账号 → Add collaborator → Copy invite link发送给被邀请人 → 接受邀请才可推送

5.抓取 pull = fetch + merge

git fetch alias branchname（这时可以查看内容决定是否合并） git merge alias/branchname（内容OK，执行合并）

git pull alias branchname（不检查，一步到位）

6.冲突问题 出现冲突需要先执行抓取命令，然后修改文件内容，确定了最终版本后，才能重新推送，解决方式与合并冲突的解决方式相同

7.跨团队协作 非团队人员访问团队远程库 → 点击fork → 非团队人员克隆fork到自己的远程库中的内容进行修改 → 推送 →

非团队人员进入自己的远程库点击Pull requests → New pull request → Create pull request（该页面可以看到自己的修改内容）→

发消息 → 团队人员进入团队远程库点击Pull requests → 审核代码：点击Commit查看提交，点击Files changed查看修改 →

回到Conversation点击Mrege pull request进行合并，确认合并时可以填写合并消息 → 将远程库修改拉去到本地

8.SSH登录 需要在家目录中执行ssh-keygen -t rsa -C [atguigu2018ybuq@aliyun.com，会生成 .ssh](mailto:atguigu2018ybuq@aliyun.com，会生成%20.ssh)目录，复制该目录下id\_rsa.pub文件中的内容

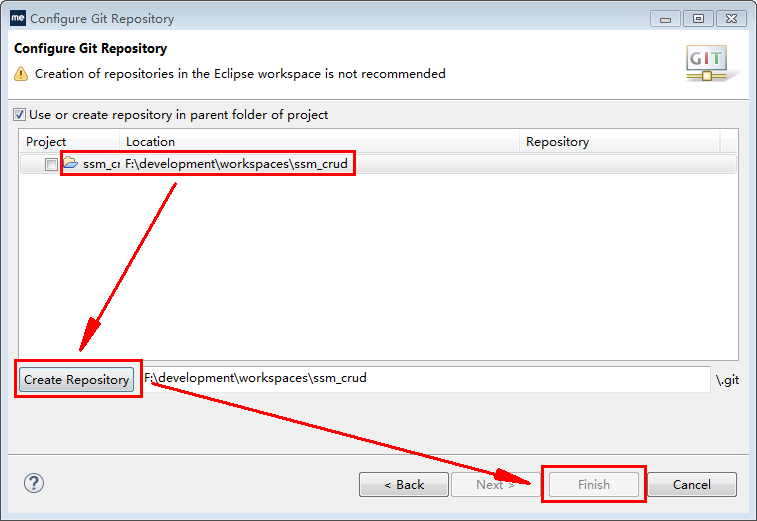
修改用户信息的SSH keys，cope内容进去即可，然后在本地库创建一个远程库的ssh地址的别名，这时推送不需要填写密码了

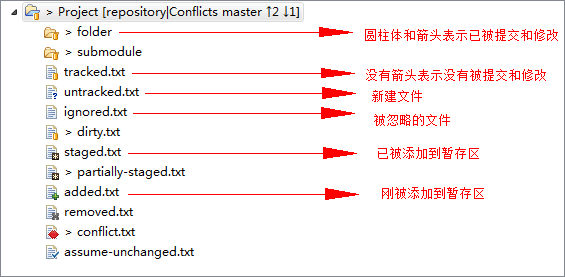
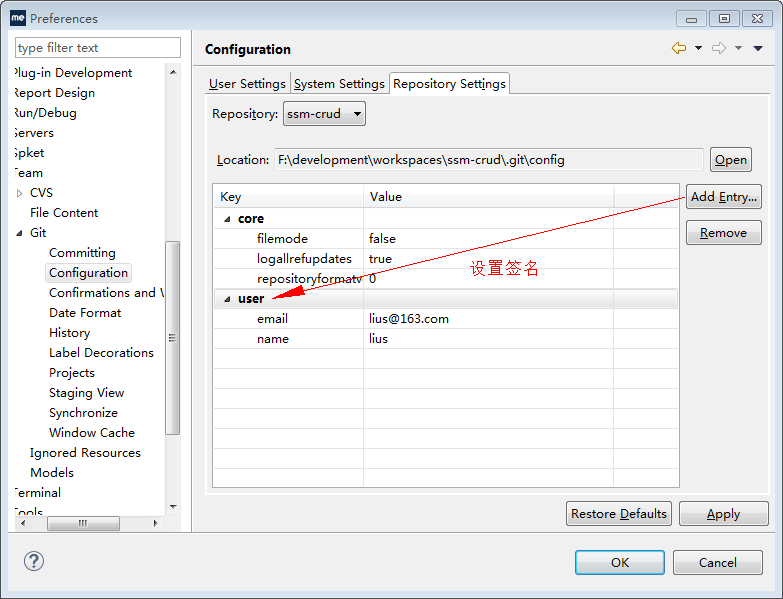
Tips：凭据管理器

## Eclipse操作Git

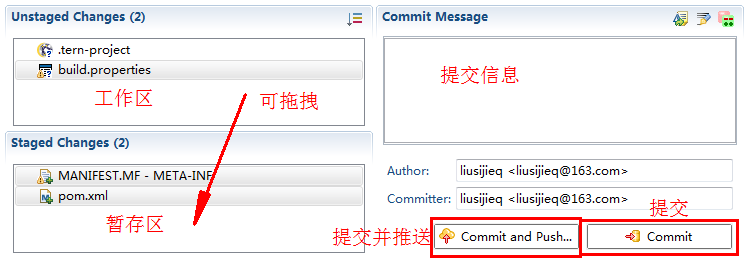
### 1.将工程初始化为本地库

工程 → 右键 → Team → Share Project → Git → →





### 2.提交

右键 → Team → Commit 

这里注意：存在很多Eclipse 特定文件，由于这些文件和开发的代码没有直接关系，并且每个版本的Eclipse的特点文件内容可能不同，这些文件不要执行提交操作

为了简化重复操作，可以通过配置文件的方式来实现

在家目录下创建文件Java.gitignore

# Compiled class file

\*.class

# Log file

\*.log

# BlueJ files

\*.ctxt

# Mobile Tools for Java (J2ME)

.mtj.tmp/

# Package Files #

\*.jar

\*.war

\*.nar

\*.ear

\*.zip

\*.tar.gz

\*.rar

# virtual machine crash logs, see http://www.java.com/en/download/help/error\_hotspot.xml

hs\_err\_pid\*

.classpath

.project

.settings

target

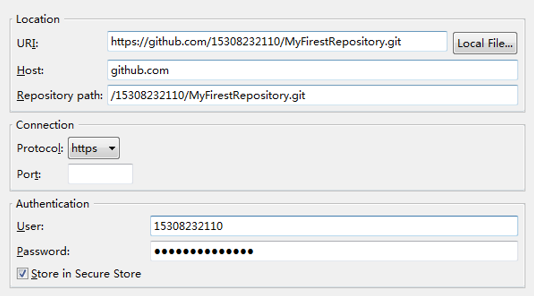
然后在 .git.config文件中引入该文件，注意使用的是“/”

[core]

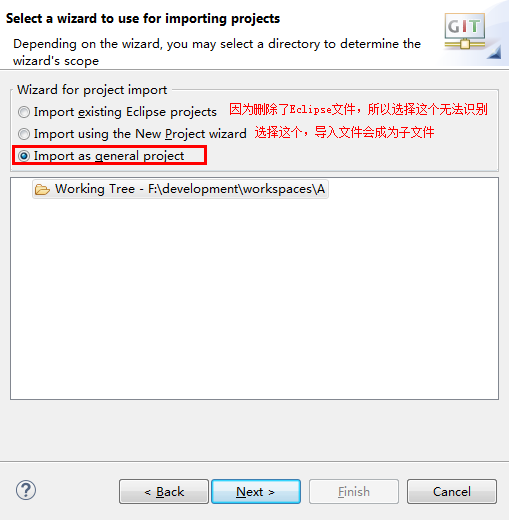
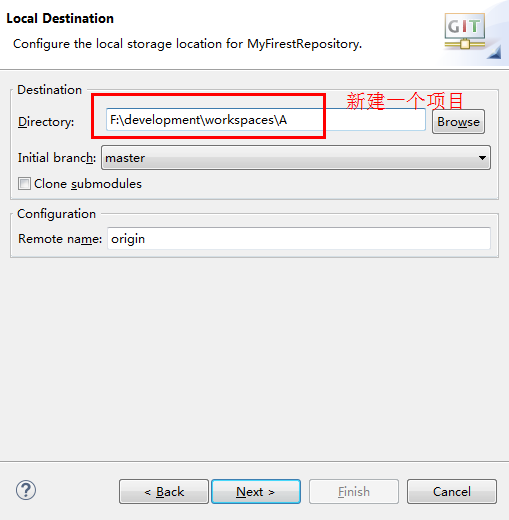
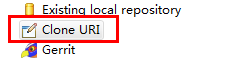
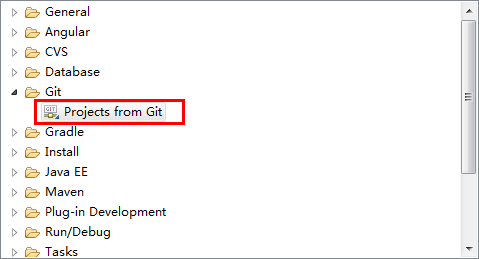
excludesfile = C:/Users/Administrator/Java.gitignore

### 3.推送

右键 → Team → Remot → Push → #git config receive.denyCurrentBranch ignore

 → Finish

### 4.克隆



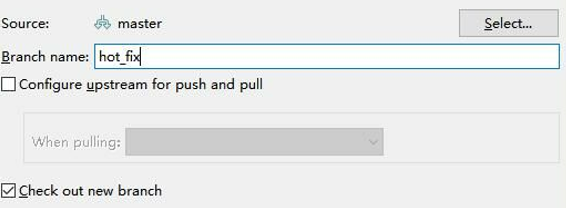
工程 → 右键 → Congigure → Convert to Maven Project

Tips：对于版本较低的Eclipse，可能需要指定文件克隆到工作区之外的目录才行

### 5.冲突

Team →  → 仍然是修改文件操作 → Team → Merge Tool（可以对比文件）→ 重新推送

### 6.分支

Team → Swith To → New Branch →  → 修改代码 → Team → Repository → Push Branch ‘hot\_fix’

Team → Swith To → Other → 选择分支，查看代码 → 切换回master分支 → 合并 → 重新推送

## Git工作流

指在项目开发过程中使用Git的方式

### 1.集中式工作流

像SVN一样，集中式工作流以中央仓库作为项目所有修改的单点实体。所有修改都提交到Master这个分支上。这种方式与SVN的主要区别就是开发人员有本地库，但Git很多特性并没有用到。

### 2.GitFlow工作流

Gitflow工作流通过为功能开发、发布准备和维护设立了独立的分支，让发布迭代过程更流畅。严格的分支模型也为大型项目提供了一些非常必要的结构。

工作流详解：

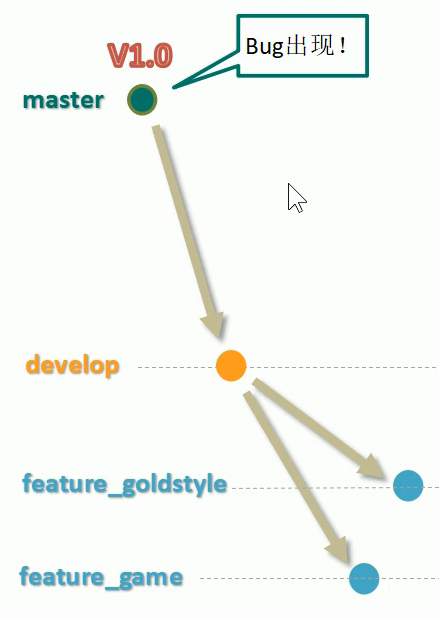
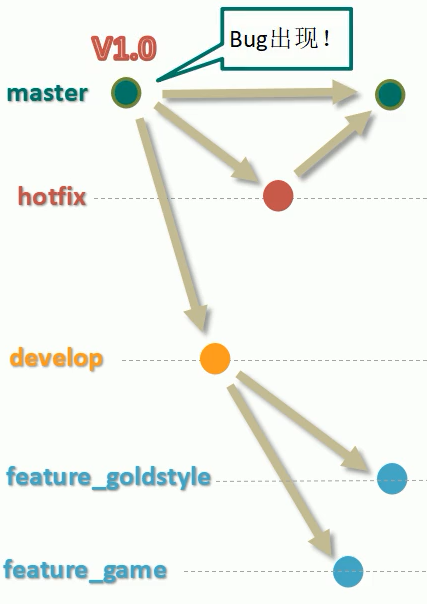
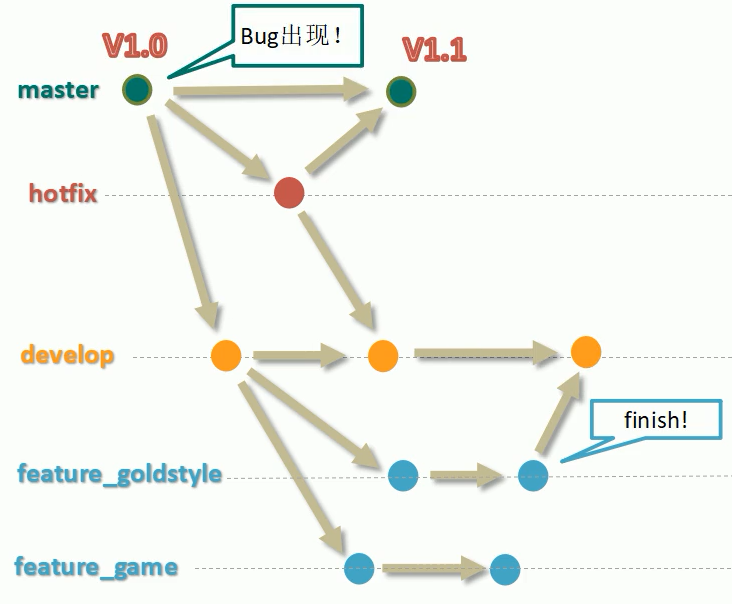
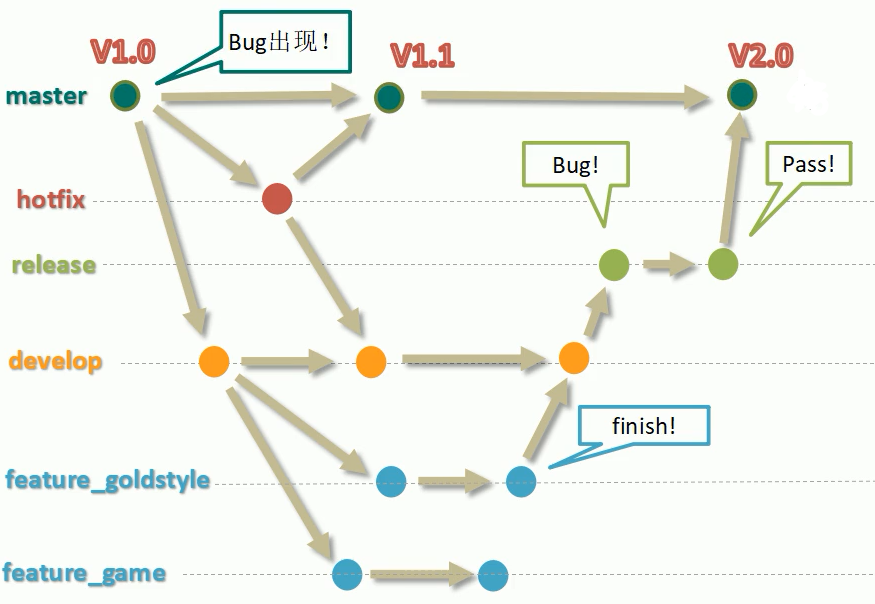
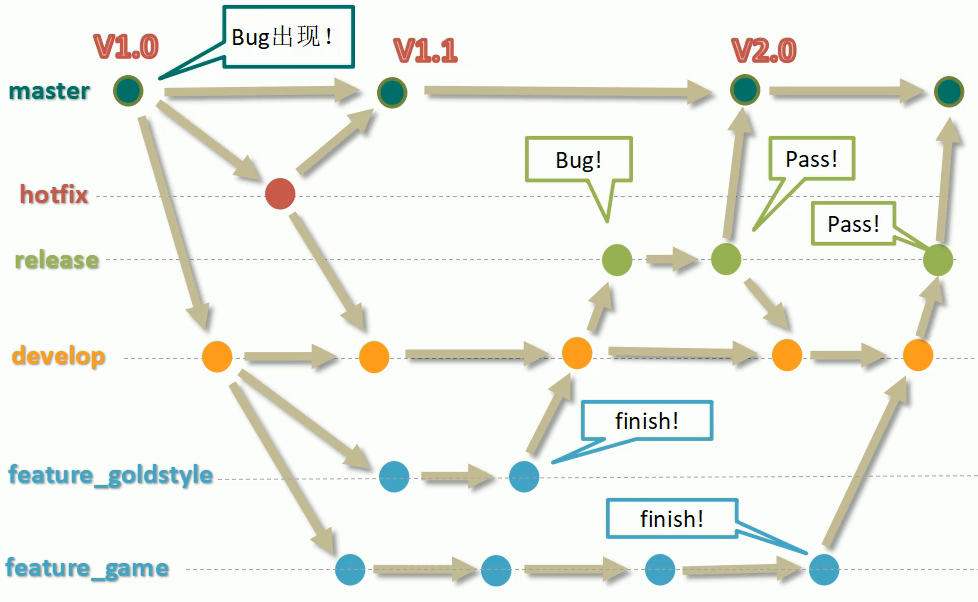
主干分支master： 主要负责管理正在运行的生产环境代码。永远保持与正在运行的生产环境完全一致。

开发分支develop： 主要负责管理正在开发过程中的代码。一般情况下应该是最新的代码。

bug修理分支hotfix： 主要负责管理生产环境下出现的紧急修复的代码。从主干分支分出，修理完毕并测试上线后，并回主干分支。并回后，视情况可以删除该分支。

准生产分支release： 进行最后阶段的集成测试。该版本上线后，会合并到主干分支。生产环境运行一段阶段较稳定后可以视情况删除。

功能分支 feature： 为了不影响较短周期的开发工作，一般把中长期开发模块，会从开发分支中独立出来。 开发完成后会合并到开发分支。

### 3.Forking 工作流

Forking工作流是在GitFlow基础上充分利用 Git的Fork和pull request的功能以达到代码审核的目的。更适合安全可靠地管理大团队的开发者，而且能接受不信任贡献者的提交。

## Gitlab服务器搭建过程

### 1.官网地址

首页：https://about.gitlab.com/

安装说明：https://about.gitlab.com/installation/

### 2.安装命令

sudo yum install -y curl policycoreutils-python openssh-server cronie

sudo lokkit -s http -s ssh

sudo yum install postfix

sudo service postfix start

sudo chkconfig postfix on

curl https://packages.gitlab.com/install/repositories/gitlab/gitlab-ee/script.rpm.sh | sudo bash

sudo EXTERNAL\_URL="http://gitlab.example.com" yum -y install gitlab-ee

实际问题：yum 安装 gitlab-ee（或ce）时，需要联网下载几百M的安装文件，非常耗时，所以应提前把所需RPM包下载好，但不要解压！

下载地址为：<https://packages.gitlab.com/gitlab/gitlab-ce/packages/el/7/gitlab-ce-10.8.2-ce.0.el7.x86_64.rpm>，该版本最好对应CentOS7

### 3.调整后的安装命令

sudo rpm -ivh /opt/gitlab-ce-10.8.2-ce.0.el7.x86\_64.rpm

sudo yum install -y curl policycoreutils-python openssh-server cronie

sudo lokkit -s http -s ssh

sudo yum install postfix

sudo service postfix start

sudo chkconfig postfix on

curl https://packages.gitlab.com/install/repositories/gitlab/gitlab-ce/script.rpm.sh | sudo bash

sudo EXTERNAL\_URL="http://gitlab.example.com" yum -y install gitlab-ce

Tips：写一个可执行脚本，执行脚本结束后需要重启

### 4.Gitlab服务操作

gitlab-ctl reconfigure 初始化配置Gitlab

gitlab-ctl start 启动Gitlab服务

gitlab-ctl stop 停止Gitlab服务

### 5.浏览器访问

访问 Linux 服务器 IP 地址即可，如果想访问 EXTERNAL\_URL 指定的域名还需要配置域名服务器或本地 hosts 文件。初次登录时需要为gitlab的root用户设置密码。

应该需要停止防火墙：service firewalld stop